{ eBook }

loggly

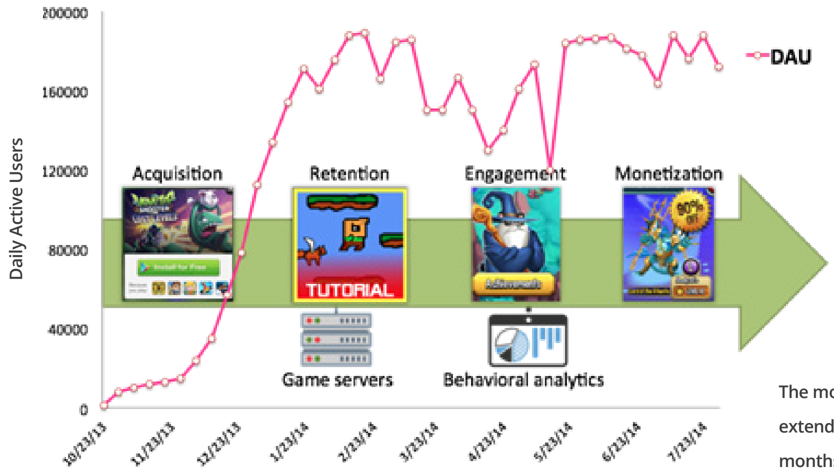
The Definitive Guide to Maintaining a Top-Grossing Game

Introduction

Game developers today maneuver in a difficult market. Major gaming platforms (consoles, PC, mobile, and web) are saturated with high-quality games offered either at low cost or completely free. As these platforms have matured, strong market leaders have emerged that routinely deliver top-grossing games. They spend hundreds of millions of dollars each year on user acquisition to keep their games dominating the charts.

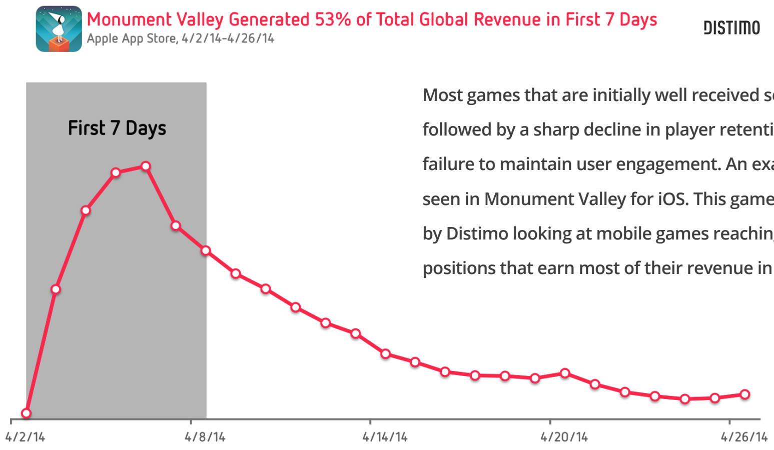
Because of the abundance of games and strong competition, the bar for providing a great player experience has been set very high for large companies and independent developers alike. This bar, however, is not impossible to clear. New companies reach the top of the mobile, PC, and console market charts frequently. What separates the big earners from the little guys is their ability to sustain that success over the long term.

What Success Looks Like



The most successful games extend their lifetimes many months or years after launch.

The Fortune of Most Games



Most games that are initially well received see a burst of excitement followed by a sharp decline in player retention, usually due to the failure to maintain user engagement. An example of this trend is seen in Monument Valley for iOS. This game was part of a study by Distimo looking at mobile games reaching top 100 app store positions that earn most of their revenue in the first week of release.

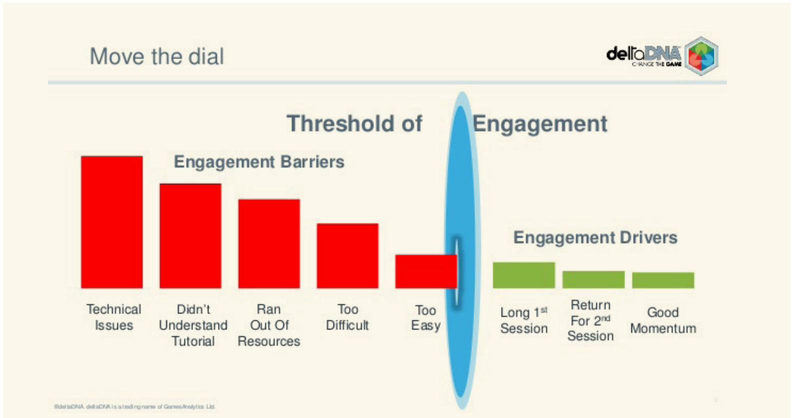
How to Keep Your Game at the Top

So what's the key to avoiding a decline in popularity and keeping your games in top ranking and earning positions? The baseline requirement is creating a fun, engaging game that your users love. But to extend a game's success into the long term, you also need:

- Ongoing **user acquisition campaigns,** targeted toward specific demographics, to bring those players to your game who will be the most likely to engage and spend in it
- A **compelling first experience** that convinces these players to engage in your game for the long term
- Constant **feature and content updates** that keep players engaged.
- **Customized gameplay** and feature offerings tailored to different players' gameplay style
- A completely **bug-free and uninterrupted** gameplay experience.

Basically, your team has to treat its game as a **live entertainment service that operates 24/7** and evolves constantly to satisfy your customers' desires

While game developers are experts at user acquisition and using player data to create good first experiences and long-term engagement, many are not prepared to deliver on a bug-free experience. Online game architectures are often complex and stressed, so technical issues like gameplay bugs, crashes, and server connection problems are common. Too many game companies fall victim to these issues, but technical problems will cause serious damage to user engagement and spending.



DeltaDNA, a leading gaming analytics company, cites technical issues as one of the top barriers to users engaging in games.

How much damage, exactly? The Ponemon Institute estimates the average direct cost related to downtime/outages at $7,900 per minute.[1] If you take into account factors such as players who never return or those who fail to convert to paying users due to these technical issues and others, the long-term cost may be even higher. According to Albert Ho, Executive Producer/Product Manager for Platform at Rumble Entertainment, "When a server goes down or when we go down, the impact on our revenue is huge. Even if there's a slight outage or things are just slow or lagging, we might see as much as a 70 to 80 percent revenue dropoff that day."

Game developers should not treat technical issues as an inevitable cost of doing business. There are concrete steps you can take to reduce downtime to minimal levels.

We have learned to expect the unexpected and always assume that something will go wrong, no matter how well we prepare.

It's very important that we have enough visibility into the systems to be able to correlate a dip in revenue to some other change, like an increase in response times, or errors from external services, and act on those problems quickly."

Chris Gander
Technical Director, Electronic Arts

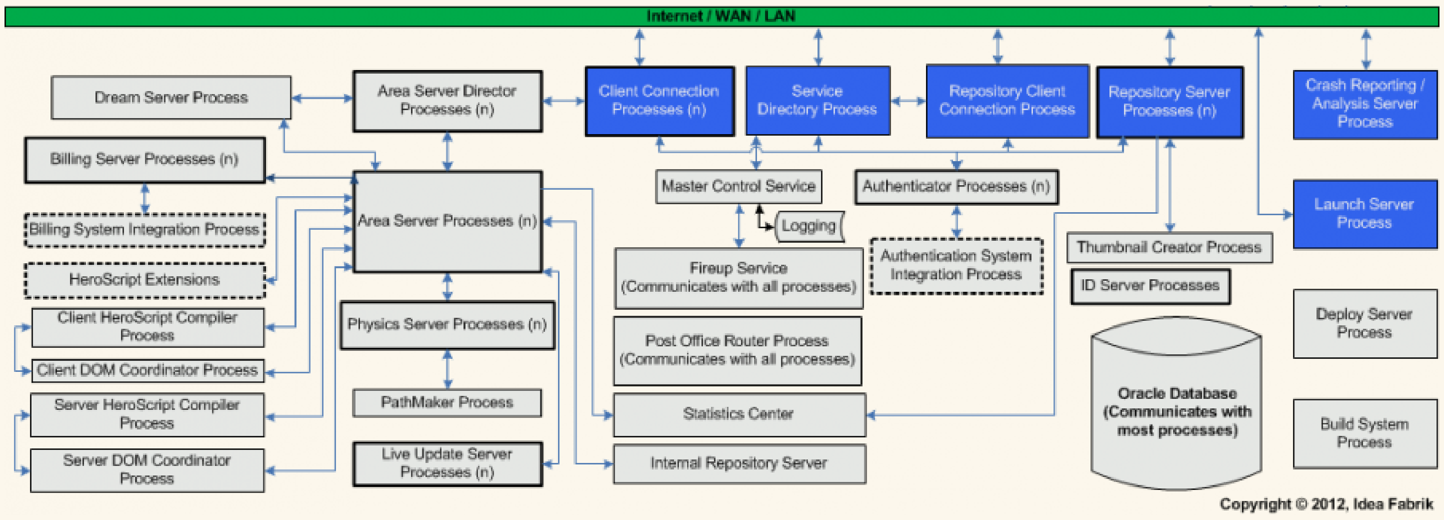
Read on to learn what you can do

[1] http://www.emersonnetworkpower.com/en-US/About/NewsRoom/NewsReleases/Pages/Emerson-Ponemon-Cost-Unplanned-Data-Center-Outages.aspx

The Technical Challenges of Live Game Operation

So what kind of distinct operational challenges do games face? Most games now must release constant content updates to stay competitive. This means that even a strictly single player game will have to contend with issues being introduced and released into production versions of the game.

This complexity is amplified many times in games that have online components. Online games often must maintain multiple servers and services for unique functions of the game. In the wild, these servers face diverse client hardware and network conditions, unpredictable and rapidly changing player loads, security challenges, ongoing updates and patches, and much more.


Internet / WAN / LAN
Dream Server Process
Area Server Director Processes (n)
Client Connection Processes (n)
Service Directory Process
Repository Client Connection Process
Repository Server Processes (n)
Crash Reporting / Analysis Server Process
Billing Server Processes (n)
Area Server Processes (n)
Master Control Service
Authenticator Processes (n)
Logging
Launch Server Process
Billing System Integration Process
HeroScript Extensions
Fireup Service (Communicates with all processes)
Authentication System Integration Process
Thumbnail Creator Process
ID Server Processes
Client HeroScript Compiler Process
Physics Server Processes (n)
Deploy Server Process
Client DOM Coordinator Process
Post Office Router Process (Communicates with all processes)
Oracle Database (Communicates with most processes)
Server HeroScript Compiler Process
PathMaker Process
Statistics Center
Build System Process
Server DOM Coordinator Process
Live Update Server Processes (n)
Internal Repository Server
Copyright © 2012, Idea Fabrik

The architecture of the popular HeroEngine MMO game server shows many moving parts.

Technical Errors and Player Impact

With this level of complexity, the types of technical errors that arise are very diverse, but they generally fall into one of four categories.

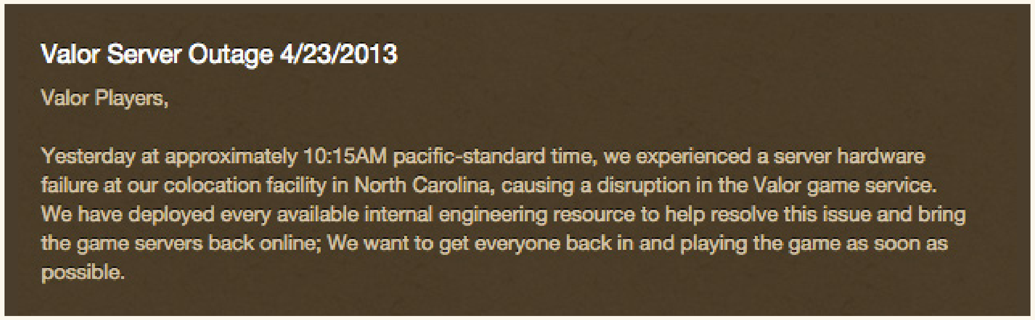
ERROR	EFFECTS
Loss of player data (items, money, XP)	**Irreversible loss of hard-earned player advancement** ▸ Heavy loss of players ▸ Massive reduction in key engagement KPIs ▸ Massive reduction in overall user lifetime value
Slow gameplay	**Significant impairment in user experience** ▸ Hard to diagnose and can last for extended periods of time ▸ Heavy reduction in key engagement KPIs ▸ Heavy reduction in overall user lifetime value
Game crashes and server downtime	**Interruption of user session** ▸ Moderate reduction in key engagement KPIs ▸ Moderate reduction in user lifetime value
Individual feature bugs	**Malfunctions of individual game features** ▸ Potentially higher drops in retention, engagement, and monetization KPIs ▸ Differential impact on players with different play styles

Common causes of technical issues include the following:

- **Database issues:** Issues with database latency or data loss or corruption can result in slow performance or loss of vital player data.
- **Scaling and load issues:** Player load on online games tends to change quickly. This requires fast scale-ups and scale-downs of resources or even fallback strategies where certain functionality is disabled in periods of high load. Load issues are a major cause of frustratingly slow gameplay and server downtime.
- **Software bugs:** Software bugs from client and server updates can introduce a variety of unexpected behaviors and disrupt certain game features, cause crashes, or even worse — result in game downtime.
- **Client OS updates:** Each device users play on — whether a mobile device, PC, or console — has a unique operating environment that changes constantly and can cause significant problems with client software or interaction with online components of the games.
- **Hardware failures:** Physical hardware failures can happen even in top cloud based IaaS/PaaS services and cause downtime and loss of data if proper redundancy measures have not been taken.

Regardless of their nature or cause, these errors waste players' time and hinder their enjoyment, often significantly shortening a player's lifetime value. Real industry examples of these issues can be seen in the next section.

Real-World Examples of Operational Issues and Their Costs

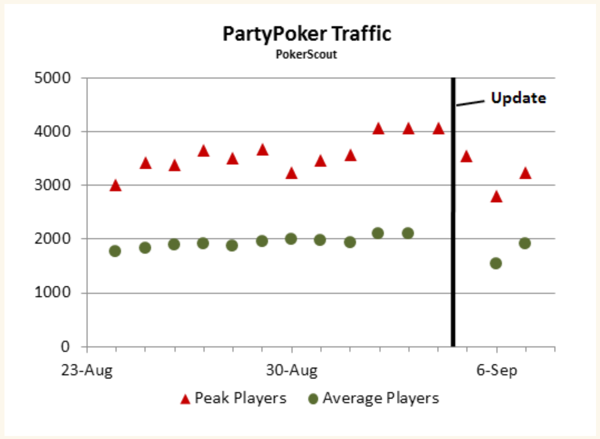


Valor Server Outage 4/23/2013

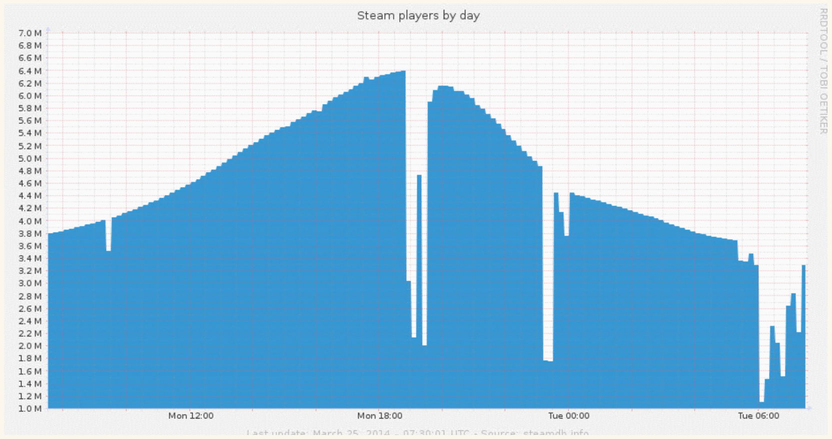
Valor Players,

Yesterday at approximately 10:15AM pacific-standard time, we experienced a server hardware failure at our colocation facility in North Carolina, causing a disruption in the Valor game service. We have deployed every available internal engineering resource to help resolve this issue and bring the game servers back online; We want to get everyone back in and playing the game as soon as possible.

The Valor outage lasted for three days while engineers worked around the clock to bring the game service back on. The company ended up giving players free boosters, as a thank you for sticking around. (Read more at http://valorgame.com/valor-now-online.)



The number of PartyPoker users dropped dramatically after the release of a problematic update. Players experienced troubles logging in and slow response times. It took weeks to solve the problem, and traffic did not swing upward until late September.



Steam had several outages in March 2014. With each outage, millions of players could not connect to the Steam network.

Examining the Impact of Errors on Different Types of Players

Technical issues are a constant problem for developers. The previous examples show how big an impact technical issues can have on engagement and revenue. To manage issues most effectively, developers need to analyze *how* those errors impact their games.

Developers who pay attention to their behavioral analytics will quickly notice that not all players are the same. Distinct types of players emerge, each with unique styles of play and characteristic spending and engagement patterns. Smart developers segment these differing play styles into distinct groups and create content and features specifically targeted to each in order to maximize their engagement and spending.

All players hate when a game doesn't work as expected and will eventually leave, but certain types have more tolerance for error than others. A child who is a hard-core grinder is more likely to forgive issues and errors than a busy, middle-aged hard-core builder with far more money to spend.

Additionally, different errors have different impacts on specific player types. Players who enjoy building things, for instance, will be much more heavily disturbed by an error in the building mechanics than a player who enjoys combat. Some player types will be far more likely to spend money than others. To minimize the cost of technical issues on your game, you need to determine which player types bring you the highest value and what errors negatively affect those players so that you can avoid them.

Player Segments

Hard-core grinders
highly engaged but spend little or no money

Hard-core builders
have a medium to high level of engagement and tend to be bigger spenders

Social peacocks
invite their friends to play but have only a low to medium level of spending

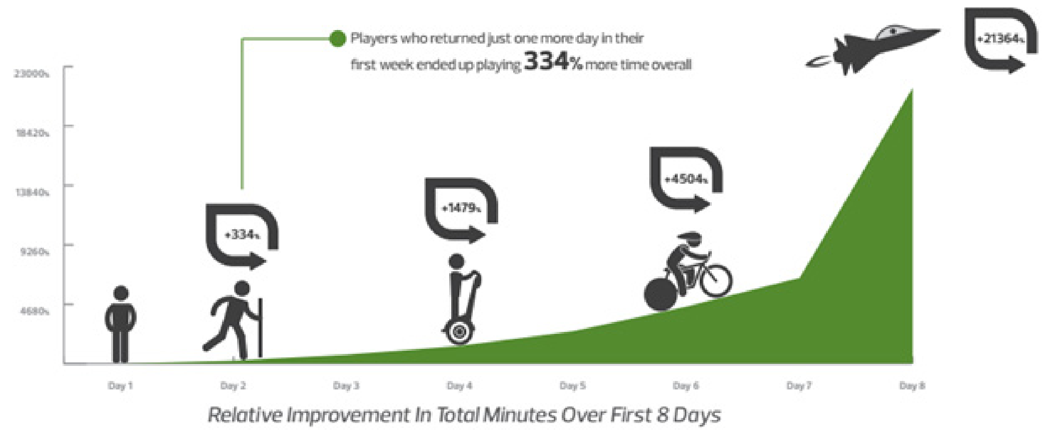
Examining the Impact of Error on Retention, Engagement, and Monetization

Technical issues affect each stage of the player lifecycle differently, so it's also important to understand how each part of the lifecycle is affected.

Player Retention

The user's first experience is crucial. Industry data from Playnomics (part of Unity Cloud) shows that the amount of hours users spend playing a game correlates strongly with the number of days they return after their initial experience. Within the first seven-day retention period, users are far less error tolerant and are far less likely to return to your game if they experience an error. This is the most crucial period to keep error free.

Correlation of Player Engagement with Retention



Credit: Playnomics

Signs of Engagement

Individual engagement
Player is heavily engaged with specific gameplay mechanics.

Social engagement
Player participates in many interactions with other players.

Viral influence
Player invites new players to join the game.

Economic behaviors
Player spends money in the game.

Player Engagement

Once you've retained players by providing an engaging and error-free first experience, you need to keep them interested. Engaged players can easily spend weeks or months playing your game, and it's in this time that they make their spending decisions.

Eliminate all possible barriers to engagement, which often include adjusting game difficulty balance, determining the best places for players to run out of resources, or providing an unnecessarily complicated tutorial. Keeping players engaged is your bridge to monetization. Errors in this phase lessen a player's interest in moving forward through the game, shorten his or her lifetime, and reduce the likelihood of conversion to a paying player.

Player Monetization

Different types of players will monetize at different points in the primary engagement period. When considering errors' effect on this process, keep in mind:

- If your first monetization event fails, you've likely lost a customer forever.
- International purchase errors are frequent, often because of improper localization of the game or implementation of payment providers.
- The more players are affected by an error, the more costly it is.
- The errors that disturb your highest value players cost you the most.
- Errors during the retention period and while making monetization decisions are more costly than others.

{ Players spend money to enhance their gameplay experience. If errors are damaging your engagement numbers, they are directly reducing the amount of money players spend within your game.

Examining Impact: Quantifying the Costs

Do you know how much money you are losing because of technical issues? It's likely a greater amount than you think.

In general, system-wide game downtime when players are unable to access your game will be your most costly error. Beyond that, however, the science of quantifying the cost of your errors gets more complex. This is because the revenue lost is more than just the direct revenue drop — it also includes revenue loss from lowered retention, loss of engaged users, failed free to paying user conversions, and wasted user acquisition spend. The chart below illustrates how cost can be calculated for each error.

Before Rumble Entertainment started using the log management provider, Loggly, technical problems could routinely reduce revenue by as much as 70 percent for several days.

TOTAL REVENUE LOST DURING AN ERROR

ACQUISITION STAGE	+	RETENTION STAGE	+	ENGAGEMENT STAGE	+	CONVERSION STAGE	+	MONETIZATION STAGE
User acquisition spending loss: (Number of users lost due to lowered retention rate)*(cost to acquire each user)		**Revenue loss in early retention period:** (Number of new users lost due to lowered retention rate)*(average user lifetime value)		**Revenue loss from engaged users:** (Number of engaged users lost in each user cohort)*(projected lifetime value of cohorts)		**Revenue loss from failed conversions:** (Number of lost conversions)*(average lifetime value of paying users)		**Direct revenue loss:** Drop in direct game revenue during game error

Quantifying cost per error can be difficult, but it is necessary. Quantifying the cost allows you determine which errors are costing you the most and which ones are the highest priority to address.

Additionally, different errors affect different user groups disproportionately. If you have categorized your users into unique groups, you will want to measure how your highest value players are affected by each error and how much these errors ultimately cost you.

Mitigating the Impact of Technical Issues

If your game will feature regular updates or online features, errors are both unavoidable and potentially costly. How costly depends on how quickly you find and solve them. The goal for any developer should be to implement tools and processes that allow these errors to be detected and fixed with minimal impact on users.

ERROR RESOLUTION STEP	WHY RESPONSE TIME CAN BE SLOW	RESPONSE TIME CAN BE REDUCED BY
Error Detection	Inadequate error & system health monitoring	Tools that monitor system health & send alerts when errors occur
Error Diagnosis	Developers unable to search logs for error data quickly	Logs centralized in a single place, easily searchable
Solution & Verification	Developers cannot completely verify solutions	Tools that give system data before & after fix is applied

System logs hold the information that's needed to detect a system error and diagnose its cause. Searching these logs manually is not feasible. It's much more effective to implement a log management solution that takes logs from your server, clients, or apps and centralizes them so your live operations team can quickly and easily search them and make sense of what's in them. Using a good logging service provides one of the *most effective* means of accelerating your response time to errors and eliminating their cost to you.

Good logging tools and practices allow you to eliminate errors before they cost you users and revenue.

What Not to Do

Response to errors is often slow because...

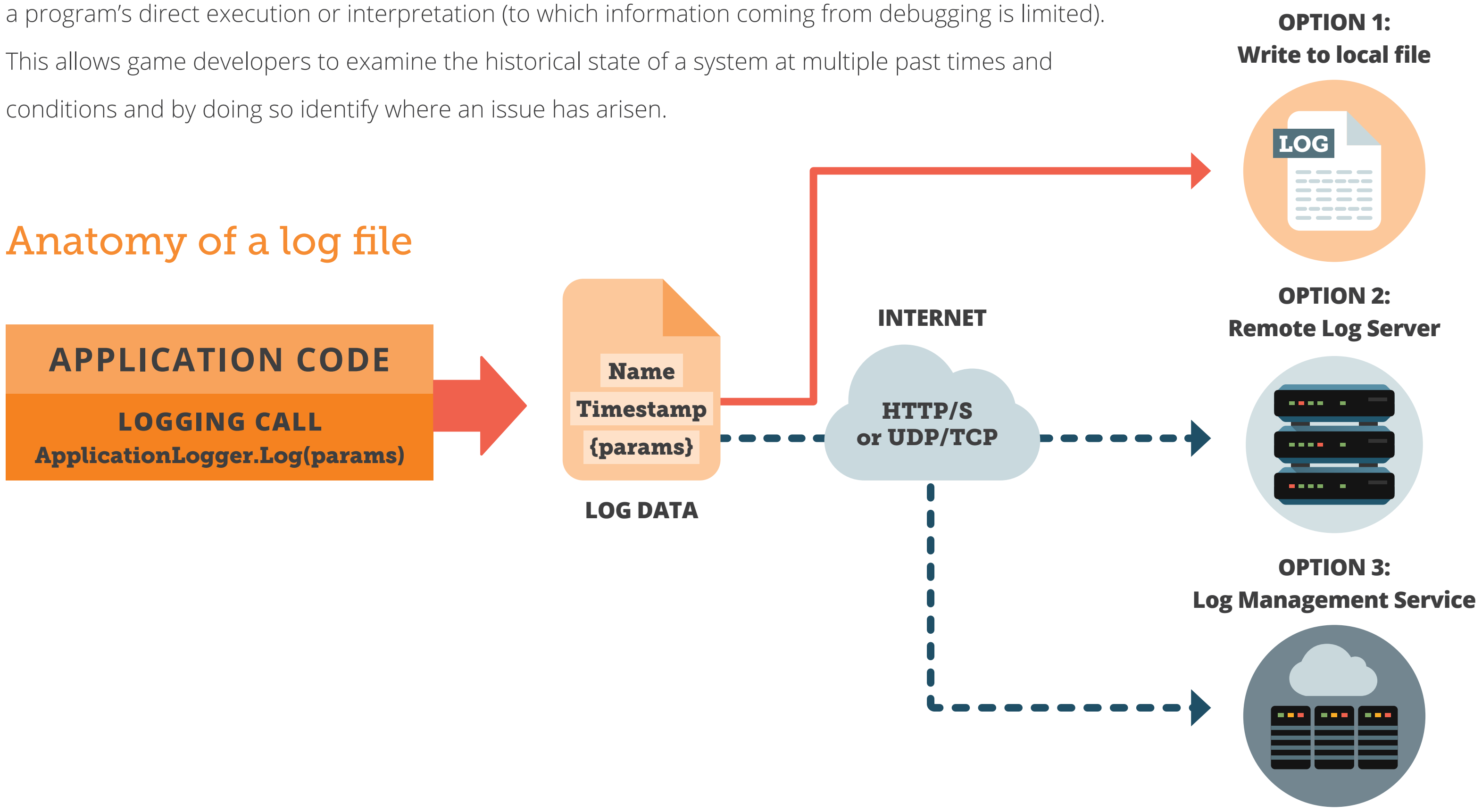
- Developers aren't monitoring their logs, application performance, or server health adequately. Many errors aren't discovered until users complain about them.
- Most games generate gigabytes or terabytes of log data, so without good tools to organize and search logs, finding the cause of errors can take many hours or even days.
- Many developers lack the tools to quickly determine if the solution has been fully effective and produces no unexpected effects.

How Is Logging Done?

In its most basic form, logging is sending data from within an application to be stored on a remote server for later examination. The data sent is usually information needed to record a snapshot of an application's state and behavior at a unique point in time.

Logging differs from debugging in that this data is recorded remotely and can be examined outside of a program's direct execution or interpretation (to which information coming from debugging is limited). This allows game developers to examine the historical state of a system at multiple past times and conditions and by doing so identify where an issue has arisen.

Anatomy of a log file


APPLICATION CODE
LOGGING CALL
ApplicationLogger.Log(params)
Name
Timestamp
{params}
LOG DATA
INTERNET
HTTP/S or UDP/TCP
OPTION 1: Write to local file
LOG
OPTION 2: Remote Log Server
OPTION 3: Log Management Service

How you collect logs will depend on the languages and frameworks your game uses. Developers most often turn to existing logging libraries, such as LOG4J (Java), LOG4NET (.NET), or Bunyan (NODE.JS), in order to send their logs to their chosen destination (console, file, or a log management system like Loggly) via RESTful or syslog protocols. Many of these platforms have the ability to send custom calls and support for multiple logging levels, which will automatically log important data like exceptions and crash logs.

GAME LANGUAGE OR ENGINE	LOGGING FRAMEWORKS	EXAMPLE CODE

Java	Log4J	`log.debug("Hello this is a debug message");`
		`log.info("Hello this is an info message");`

Swift	CocoaLumberjack, XCGLogging	`DDLogError(@"Broken sprocket detected!");`
		`log.debug("A debug message")`

C#	Log4net	`log.Error("This is my error", ex);`

Node.js	Bunyon, Winston	`Winston: log.warn({lang: 'fr'}, 'au revoir');`
		`Bunyon: category1.info('logging from your IoC container-based logger');`

Unity (C#/C++)	Robop Logger	`Log.dbg("Log message", params)`

What to Log?

The general wisdom in tech is to "log everything," which roughly translates to "log everything important." But what is important? Every game has a unique architecture so the answer will always be different. Generally, you should log information about:

- **The performance and health of your game's servers,** including such measures as CPU usage and memory allocation.
- **Your server-side application's performance and behavior,** including errors and exceptions, response times, and synch errors. If your game has real-time elements or PvP play, log key client and server events surrounding this; PvP interaction issues are a major source of user frustration.
    - **Your database operation.** Monitor the general health of database servers and log events that have to do with data transactions, data consistency, and data integrity. Any loss of data causes huge user retention issues.
    - **Your client's behavior** (game crashes, reloads, or client-side exceptions), the state of the device it's being used on, client-side network conditions, and code that interacts with your server application. Many times you can match up this client-side data with back-end problems.

If your game is client-only, you should still log crashes, exceptions, and select information on the application's behavior. This will help you figure out quickly what's causing those pesky problems in your application.

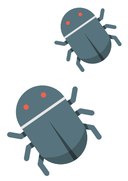
> *Measurement beats intuition every time. My own intuition is usually wrong."*
>
> **Randy Shoup**
>
> *CTO Kixeye*
>
> (Developer of the successful Backyard Monsters, Battle Pirates, and Vega Conflict titles)

How Logging Complements Debugging

One common debate over addressing game errors is whether to use logging or debugging. In reality, both are tools that should be used together as the developer solving a problem sees fit. However, developers should never go without logging their live games because it's the only way to get crucial data such as:

- What happened before and after the error being examined
- The state of the system environment surrounding any event or piece of data
- Insight into unexpected use-cases, client-side conditions, and server-side conditions that happened in live gameplay and that are extremely hard to reproduce on development servers
- A comparison of client and server conditions surrounding a unique time or data point

Once your logs point you to the causes of an error, debugging gets a lot easier. But developers without a log management system must do a lot of guesswork, wasting valuable time, development resources, and sanity.



The Challenges of Logging

Online games generate gigabytes or, in the case of games with very high DAU, terabytes of log data each week. And these are across multiple servers and clients. Imagine that an error event occurs. As the developer, you need to find out what happened. What do you do? Without good tools to organize and search through your logs, you must use old-fashioned, error-prone methods like grepping, which will take you a very long time. **Isolating the cause of errors manually can take days — during which money is trickling through your fingers with every passing second.**

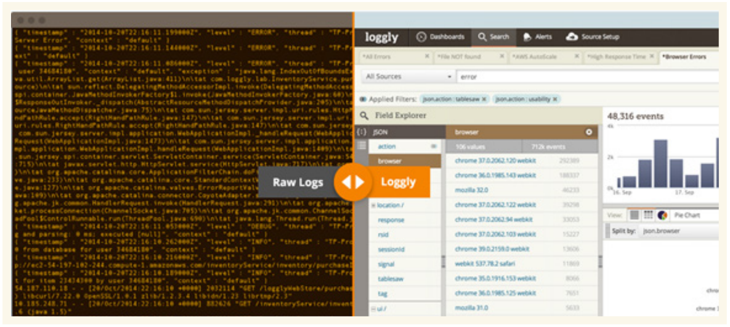
You could build a log management system yourself. But in-house solutions are rarely cost effective. Developers that aggregate their own logs must create custom data models for storing and organizing logs, and then implement custom tools for searching them. The process is long and the system is expensive to maintain. **For a lot of companies, maintaining an in-house system requires a lot of developer time and cloud/hardware expenses and often does not yield the level of log data visibility that developers need.** Nor does it offer the speed that mature log management services like Loggly provide.

{ Many developers find that they aren't able to develop an adequate system for log management. Using a system like Loggly takes the technical challenges of log management off your hands so you can focus on using the logs to solve technical issues and get your game running — and earning money — again.

Log Management Solutions

A third-party log management solution such as Loggly provides you with all the functionality you need to collect, search, and understand your log data.

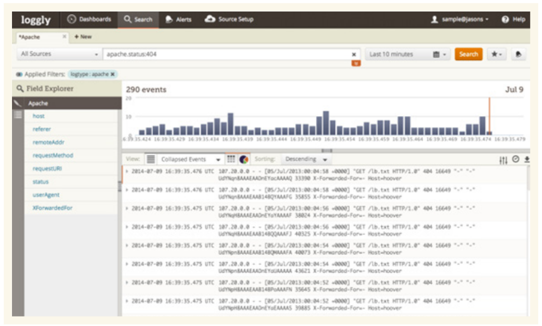
- ***Aggregation and storage is efficient and automatic.*** Logs are collected for you so you no longer need to store them in-house. They can be organized by the component that sent them (e.g., JSON.responsetime, Java garbage collector, or SYSLOG.error).



View Larger

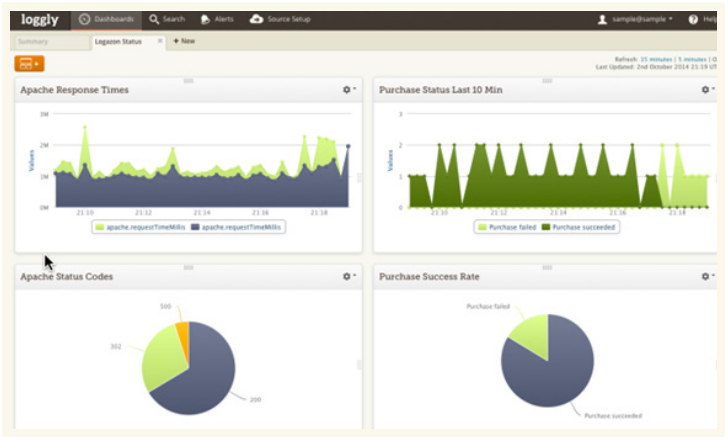
This allows you to isolate errors and compare systems, helping you find individual logs, filter out non-relevant ones, and figure out what's going on faster if there's a problem.

- ***Searching is a straightforward process.*** Log management solutions let you search all logs with custom parameters, allowing you to quickly find the specific data you need to diagnose the cause of your issues.



View Larger

- ***Visualization simplifies troubleshooting and trend analysis.*** You can represent your log data in different types of charts to help your developers easily spot errors as well as performance trends.



View Larger

Log Management Spotlight: Rumble Entertainment

Rumble Entertainment, developer of **KingsRoad**, originally relied on in-house log management. The costs with this solution were over six figures a month to manage the logs, and the company was still finding out about serious issues within their games from their players. Rumble's development team made the determination that they needed an external log solution that gave them better visibility and that they didn't have to maintain in-house.

Rumble investigated several log management systems but soon settled on Loggly for its key advantages:

- Ease of setting up the REST API. Rumble game logs were being generated within one hour.
- Focus on customer self-sufficiency. Rumble was able to set everything up itself and was running at full power within two weeks.
- Undifferentiated heavy lifting. Getting good at log management wasn't going to make Rumble better game developers, so taking that load off was enormously helpful.

"*Log management is critical for running an operation service like a game. If we didn't have Loggly, there would be days where our revenue would be affected by as much as 70 percent.*"

Albert Ho
Executive Producer/Product Manager for Platform, Rumble Entertainment

The system was up and running within a day. Rumble soon saw results.

- Doubled log management capacity and tripled data retention time — all with monthly costs reduced by more than 75 percent
- Debugging time cut from days to mere minutes
- Improved vendor management, planning, and customer service

The entire staff now relies on Loggly to keep Rumble's games running smoothly.

Log Management Strategies to Maintain Uptime and Avoid Revenue Loss

A good log management implementation provides you a detailed picture of your system's operation, which helps you identify issues quickly, before they interrupt gameplay.

However, what constitutes a "good log management implementation" and how does it help you? Below we examine technical and process strategies for implementing log management in a way that puts your team in a position to find and respond to most issues quickly.

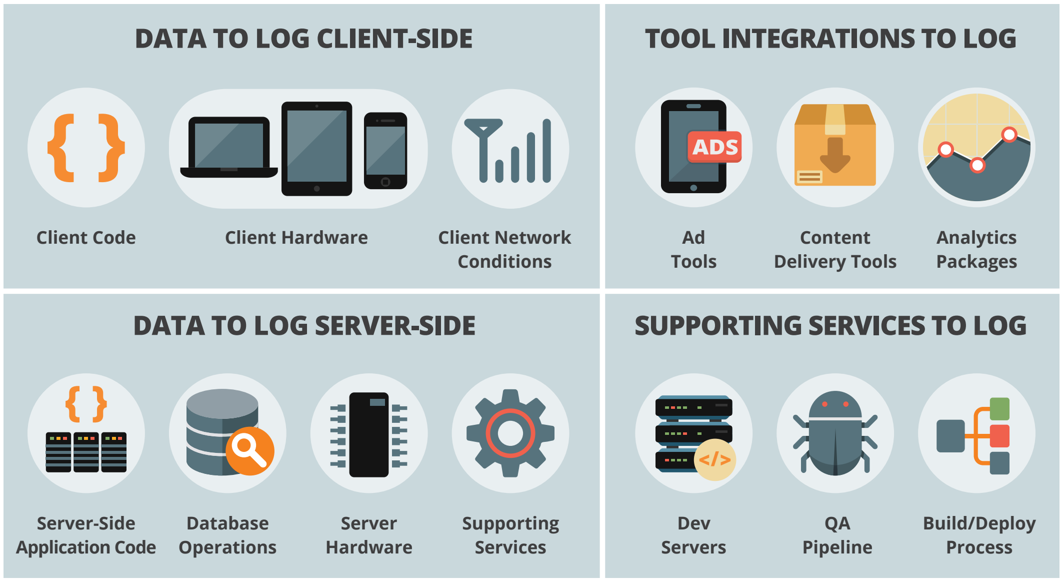
Log the Right Things

Most games, especially online games, are architecturally complex with multiple clients, computationally intensive operations, several back-end applications hosted across multiple logically distinct servers, and unpredictable traffic. This complexity provides multiple vectors for errors to arise.

The best way to detect errors within a complex system is to log the behavior and performance of most of the game's architecture. Not every detail will be important, but developers want to log data that is key to indicating problems in operation such as hardware performance stats, network status, database and message queue data, network error codes, and software exceptions/crashes/errors.

{ In a recent survey of 214 DevOps professionals, the majority of respondents reported that more than 70% of the time spent solving operational issues is spent finding their root cause.

The diagram below provides some general guidelines for the types of logging game developers should do.



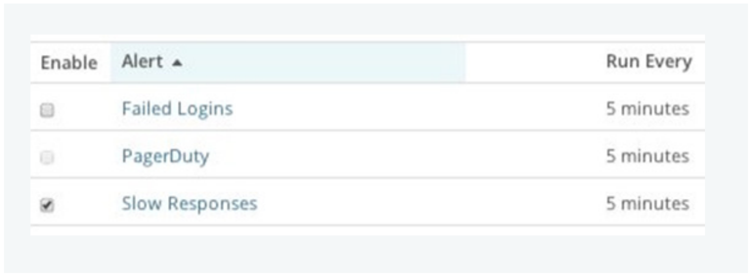
Define Healthy Performance and Set Internal Service Level Agreements (SLAs)

Once you are centralizing log data in a log management platform, you will want to sit down with the teams that manage each major component of your game and develop an explicit picture of what healthy system-wide performance looks like in your game. This will often include things like:

- **Network performance** — Maximum response times, error code limits
- **Server hardware performance** — CPU consumption, memory allocation, limits on resources dedicated to individual processes
- **Software performance** — Error messages, execution times, asset load times
- **Customer KPIs** — Apdex scores

Set Custom Alerts to Detect Errors and SLA Violations

Once healthy performance is defined, you have an explicit gauge for when system behavior deviates from healthy operation. However, your team needs to know when this happens so that they can investigate and respond immediately. Some log management tools have functionality to send custom alerts. It is a best practice to have a systematic way to let the appropriate people know when things deviate from the norm (example shown below).

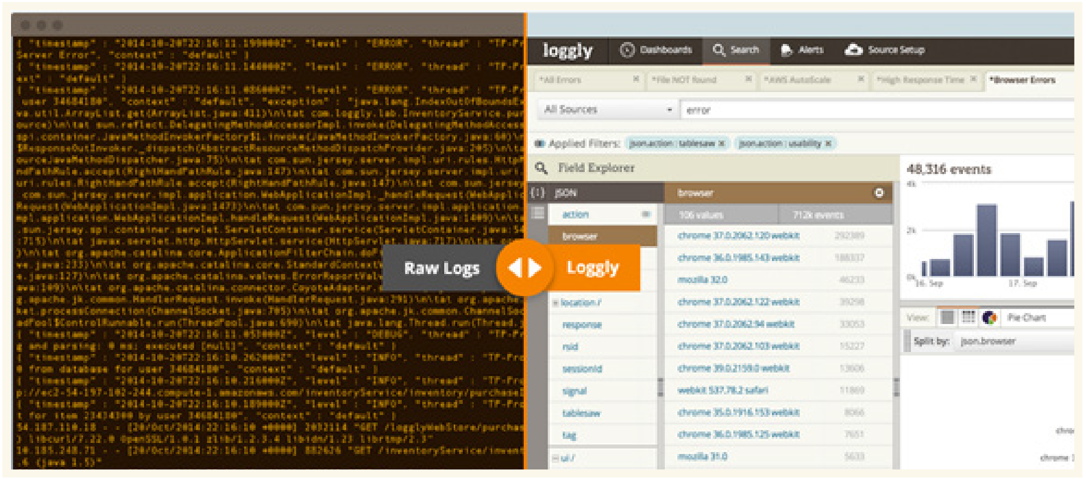


Enable	Alert ▲	Run Every
☐	Failed Logins	5 minutes
☐	PagerDuty	5 minutes
☑	Slow Responses	5 minutes

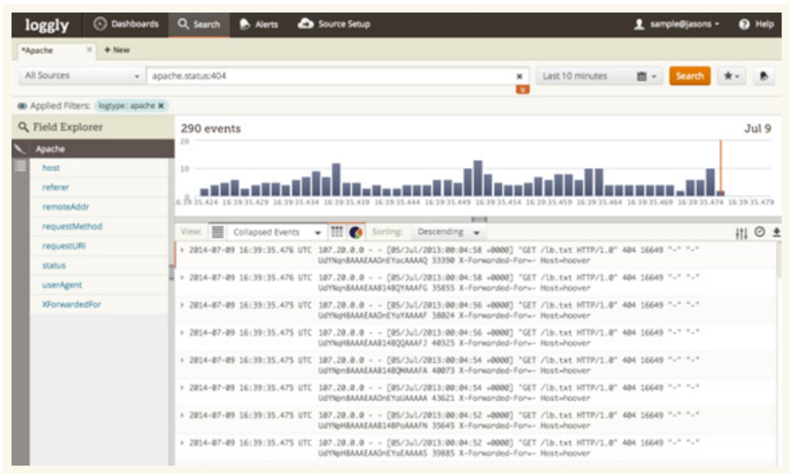
What Is the Apdex Score?

Apdex (Application Performance Index) is an open standard defining a method to report, benchmark, and track application performance. Apdex is a numerical measure of user satisfaction with the performance of enterprise applications. It translates many individual response times, measured at the user-task level, into a single number.

Organize and Display Log Data in Meaningful Ways



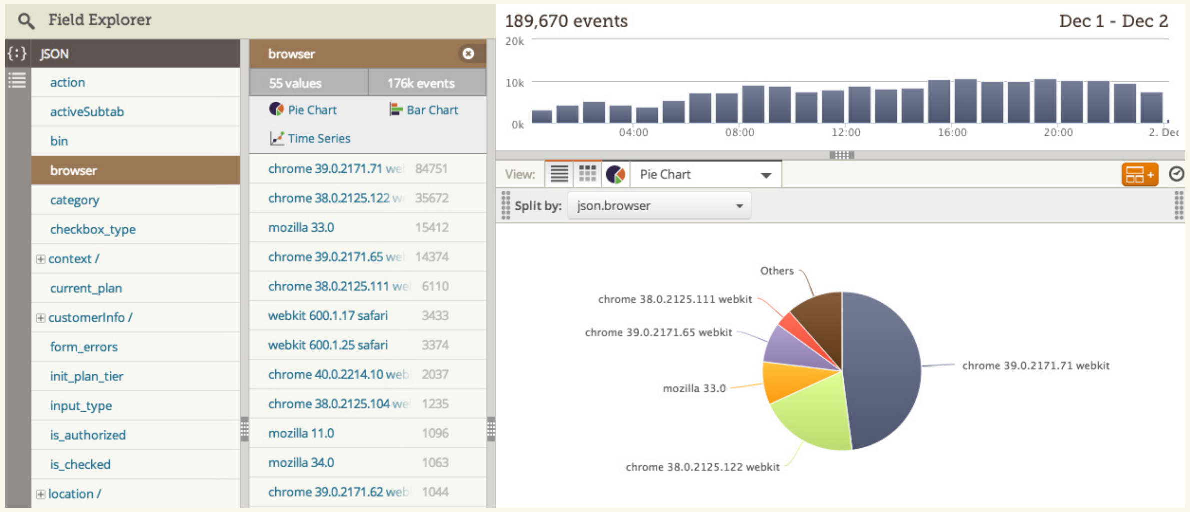
Raw log data will provide answers but data is normally verbose and hard to read. Your team can interpret log data much faster and diagnose issues more quickly when it is shown in a well-organized fashion. A log management solution like Loggly will automatically organize your logs for you according to the component that sent it and the JSON structure within the message, allowing you to quickly pull up the log data you need. When you are thinking about what to log, it's helpful to create a taxonomy that maps to how your team would troubleshoot a problem.



```
2014-07-09 16:39:35.476 UTC 107.20.0.0 - - [05/Jul/2013:00:04:58 +0000] "GET /lb.txt HTTP/1.0" 404 16649 "-" "-"
                            UdYNqnBAAAEAADnEYocAAAAQ 33390 X-Forwarded-For=- Host=hoover
2014-07-09 16:39:35.476 UTC 107.20.0.0 - - [05/Jul/2013:00:04:58 +0000] "GET /lb.txt HTTP/1.0" 404 16649 "-" "-"
                            UdYNqnBAAAEAAB148QYAAAFG 35855 X-Forwarded-For=- Host=hoover
2014-07-09 16:39:35.475 UTC 107.20.0.0 - - [05/Jul/2013:00:04:56 +0000] "GET /lb.txt HTTP/1.0" 404 16649 "-" "-"
                            UdYNqHBAAAEAADnEYoYAAAAF 38024 X-Forwarded-For=- Host=hoover
2014-07-09 16:39:35.475 UTC 107.20.0.0 - - [05/Jul/2013:00:04:56 +0000] "GET /lb.txt HTTP/1.0" 404 16649 "-" "-"
                            UdYNqHBAAAEAAB148QQAAAFJ 40325 X-Forwarded-For=- Host=hoover
2014-07-09 16:39:35.475 UTC 107.20.0.0 - - [05/Jul/2013:00:04:54 +0000] "GET /lb.txt HTTP/1.0" 404 16649 "-" "-"
                            UdYNpnBAAAEAAB148QMAAAFA 40073 X-Forwarded-For=- Host=hoover
2014-07-09 16:39:35.475 UTC 107.20.0.0 - - [05/Jul/2013:00:04:54 +0000] "GET /lb.txt HTTP/1.0" 404 16649 "-" "-"
                            UdYNpnBAAAEAADnEYoUAAAAA 43621 X-Forwarded-For=- Host=hoover
2014-07-09 16:39:35.475 UTC 107.20.0.0 - - [05/Jul/2013:00:04:52 +0000] "GET /lb.txt HTTP/1.0" 404 16649 "-" "-"
                            UdYNpHBAAAEAAB148PoAAAFN 35645 X-Forwarded-For=- Host=hoover
2014-07-09 16:39:35.475 UTC 107.20.0.0 - - [05/Jul/2013:00:04:52 +0000] "GET /lb.txt HTTP/1.0" 404 16649 "-" "-"
                            UdYNpHBAAAEAADnEYoEAAAAS 39885 X-Forwarded-For=- Host=hoover
```

Further, a log management solution like Loggly lets you visualize data in graphs in order to quickly identify trends and then save these graphs for later viewing in dashboards. Creating dashboards with visualizations of trends important to your system's health and operation will allow your team to quickly evaluate system health and any anomalies that exist within the system.

Spotlight: Loggly Dynamic Field Explorer



Fields: action, activeSubtab, bin, browser, category, checkbox_type, context /, current_plan, customerInfo /, form_errors, init_plan_tier, input_type, is_authorized, is_checked, location /

browser	
55 values	176k events
chrome 39.0.2171.71 we	84751
chrome 38.0.2125.122 w	35672
mozilla 33.0	15412
chrome 39.0.2171.65 we	14374
chrome 38.0.2125.111 we	6110
webkit 600.1.17 safari	3433
webkit 600.1.25 safari	3374
chrome 40.0.2214.10 web	2037
chrome 38.0.2125.104 we	1235
mozilla 11.0	1096
mozilla 34.0	1063
chrome 39.0.2171.62 web	1044

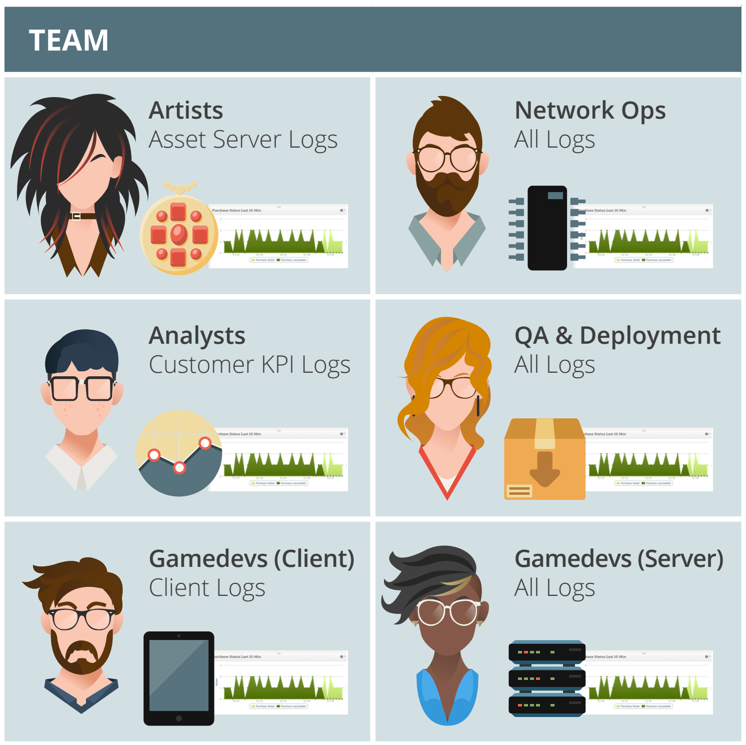
Log analysis can be a real pain if you have to write query after query to find clues that are scattered throughout your logs like hidden treasure. Loggly Dynamic Field Explorer™ is a new, real-time log management user experience that fundamentally changes how you perform operational troubleshooting. It organizes your logs by their inherent structure and includes a complete, continuously refreshed catalog of all of your fields along with the frequency of specific field values.

Loggly Dynamic Field Explorer enables you to search smarter, spot anomalies instantly, and solve problems faster by:

- **Surfacing indicators** of where to look for an operational problem or bug before you perform a single search
- **Supercharging your manual searches** with more context around the problem
- **Providing event counts** that expose the most common values as well as anomalies for every field in your log data
- Giving you a way to **immediately quantify** a problem's impact on your users

Provide the Relevant Log Data to Team Members

Each member of your team will be responsible for different parts of the system. You want to provide your team with custom views of log data that are important to their role and the parts of the system for which they're responsible. In this way, members will have a constant view of how their assigned systems are performing and can easily communicate with other team members if things go wrong.



Log Before and After Major Changes and Deployments

When updates are pushed live to any piece of the game's architecture, system state should be checked before and after to ensure no issues have been introduced. Monitoring log metrics through alerts and dashboards is an efficient way to do this.

Log Interactions with Third-Party Tools

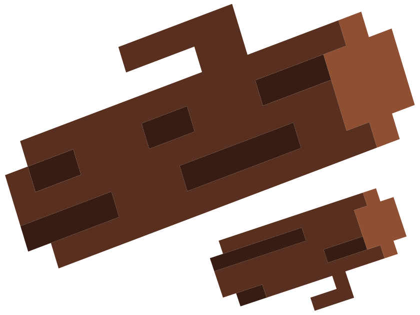
Third-party software can introduce bugs and other technical issues to your game. Be sure to log the performance of all third-party tools you use, including response times from their servers.

Identify Long-Term Trends

In the process of identifying errors, having a history of system operation will help your team see which errors are occurring most frequently, which are the most severe, and which are the most costly. This information, when retained and used in your planning exercises, will help you identify long-term performance trends and make optimizations to your game's architecture over time.

Implementing a Log Management Service

An effective log management strategy enables you to catch serious issues faster and save thousands or millions of dollars in revenue. The next question is: How would you implement a third-party log management solution like Loggly? If you're reading this eBook, it's likely that you're either beginning to implement a log management system for your games or have an existing logging infrastructure and are looking for a better way to gain insight from your log data.



If you don't already have a logging system, consider logging both your front-and back-end components. Implementing full system-wide logging for your game may take longer and more resources than you have available. However, you can still realize significant benefits by sending the logs surrounding your key issues to a log management system like Loggly.

If you already have logging in place, Loggly requires no custom agents. It's easy to change the logging endpoints of your current logging code to Loggly.

In the following section, you'll find a guide for implementing Loggly for developers who are implementing logging as well as those who already have an existing logging system.

Client-Side Logging

For client-side logging there are two options.

- **REST API:** Loggly has a detailed REST API that allows logs to be sent directly to Loggly. If you already implemented client-side logging RESTfully or want to implement REST logging, you can get started **here**. Loggly's RESTful APIs can be used with common game engines such as Unity3D, Unreal Engine, CryEngine, Corona SDK, and PhoneGap.
- **Language-specific libraries:** Loggly also offers many language specific libraries and integrations with common client-side logging libraries like Log4j, angular.js, and CocoaLumberjack for iOS. Information on using these language specific logging libraries can be found **here**.

Back-End Logging

- **Syslog:** The most common method of collecting logs from the back end is syslog. Loggly has several robust ways of consuming logs, most of which can integrate with existing logging implementations by changing endpoints. Documentation for syslog logging can be found **here**.

- **Custom tech stacks:** Each back end is built with a custom tech stack, and Loggly has many options for the different technologies used to build modern gaming back ends such as AWS CloudTrail, Heroku, Amazon S3, Logstash, MySQL database languages, and more, all of which can be found **here**.

As soon as you start sending your log data to Loggly, you will be able to view it and organize it. With only a few hours' work, you can have a higher level of visibility within your game's architecture than you ever had previously!

Parting Thoughts

Running a successful game relies on more than just gameplay. Operational failures have taken down even some of the biggest brands — and they can take down your game, too.

There are a host of technical issues that can undermine the player experience, even in the best-designed game. Using log management as a tool for maintaining consistently great gameplay protects your hard-won revenue. It also amplifies the impact of your investments in customer acquisition by eliminating technical issues as barriers to monetization. Making the investment in DevOps — and the instrumentation and tools that support the team's activities — is a winning strategy for any game developer.

The benefits are numerous:

- The whole team will **consider the system's health** when analyzing user behavior and pushing new content and features.
- You'll **keep release cycles on track** and ensure high-quality player experience even as your game evolves.
- You'll be able to **react faster** when the inevitable technical issues crop up.
- Using your system data effectively will **free up developer time** and make all your marketing spend more effective.

Loggly is free to set up and use, so why not get started right now?

TRY LOGGLY FOR FREE:
loggly.com/trial

About Loggly

Loggly is the world's most popular cloud-based, enterprise-class log management solution, used by more than 5,000 happy customers to effortlessly spot problems in real time, easily pinpoint root causes, and resolve issues faster to ensure application success. Founded in 2009 and based in San Francisco, the company is backed by Trinity Ventures, True Ventures, Matrix Partners, Cisco, Data Collective Venture Capital, and others.

Visit the Loggly website: loggly.com

Representative Loggly Game Developer Customers

Appendix: Directing Your Logging Endpoints to Loggly

The documentation on how to direct your endpoints to Loggly is available here:

- **Logging from software applications (Flash, PHP, Java, SQL, etc.)**

  https://www.loggly.com/docs/logging-from-applications

- **Logging from systems (Linux, Syslog)**

  https://www.loggly.com/docs/sending-logs-unixlinux-system-setup

- **Logging via RESTful HTTP/S communication**

  https://www.loggly.com/docs/sending-logs-unixlinux-system-setup

{ eBook }

loggly

TRY LOGGLY FOR FREE:
loggly.com/trial